**opencv 利用傅里叶变换获取低频和高频部分图像-07**

**1. 效果**



**2. 代码**

**# -\*- coding: utf-8 -\*-**

**import cv2**

**import cv2 as cv**

**import numpy as np**

**from matplotlib import pyplot as plt**

**def filter\_high\_f(fshift, radius\_ratio):**

**"""**

**过滤掉除了中心区域外的高频信息**

**"""**

**# 1, 生成圆形过滤器, 圆内值1, 其他部分为0的过滤器, 过滤**

**template = np.zeros(fshift.shape, np.uint8)**

**crow, ccol = int(fshift.shape[0] / 2), int(fshift.shape[1] / 2) # 圆心**

**radius = int(radius\_ratio \* img.shape[0] / 2)**

**if len(img.shape) == 3:**

**cv2.circle(template, (crow, ccol), radius, (1, 1, 1), -1)**

**else:**

**cv2.circle(template, (crow, ccol), radius, 1, -1)**

**# 2, 过滤掉除了中心区域外的高频信息**

**return template \* fshift**

**def filter\_low\_f(fshift, radius\_ratio):**

**"""**

**去除中心区域低频信息**

**"""**

**# 1 生成圆形过滤器, 圆内值0, 其他部分为1的过滤器, 过滤**

**filter\_img = np.ones(fshift.shape, np.uint8)**

**crow, col = int(fshift.shape[0] / 2), int(fshift.shape[1] / 2)**

**radius = int(radius\_ratio \* img.shape[0] / 2)**

**if len(img.shape) == 3:**

**cv2.circle(filter\_img, (crow, col), radius, (0, 0, 0), -1)**

**else:**

**cv2.circle(filter\_img, (crow, col), radius, 0, -1)**

**# 2 过滤中心低频部分的信息**

**return filter\_img \* fshift**

**def ifft(fshift):**

**"""**

**傅里叶逆变换**

**"""**

**ishift = np.fft.ifftshift(fshift) # 把低频部分sift回左上角**

**iimg = np.fft.ifftn(ishift) # 出来的是复数，无法显示**

**iimg = np.abs(iimg) # 返回复数的模**

**return iimg**

**def get\_low\_high\_f(img, radius\_ratio):**

**"""**

**获取低频和高频部分图像**

**"""**

**# 傅里叶变换**

**# np.fft.fftn**

**f = np.fft.fftn(img) # Compute the N-dimensional discrete Fourier Transform. 零频率分量位于频谱图像的左上角**

**fshift = np.fft.fftshift(f) # 零频率分量会被移到频域图像的中心位置，即低频**

**# 获取低频和高频部分**

**hight\_parts\_fshift = filter\_low\_f(fshift.copy(), radius\_ratio=radius\_ratio) # 过滤掉中心低频**

**low\_parts\_fshift = filter\_high\_f(fshift.copy(), radius\_ratio=radius\_ratio)**

**low\_parts\_img = ifft(low\_parts\_fshift) # 先sift回来，再反傅里叶变换**

**high\_parts\_img = ifft(hight\_parts\_fshift)**

**# 显示原始图像和高通滤波处理图像**

**img\_new\_low = (low\_parts\_img - np.amin(low\_parts\_img)) / (np.amax(low\_parts\_img) - np.amin(low\_parts\_img) + 0.00001)**

**img\_new\_high = (high\_parts\_img - np.amin(high\_parts\_img) + 0.00001) / (np.amax(high\_parts\_img) - np.amin(high\_parts\_img) + 0.00001)**

**# uint8**

**img\_new\_low = np.array(img\_new\_low\*255, np.uint8)**

**img\_new\_high = np.array(img\_new\_high \* 255, np.uint8)**

**return img\_new\_low, img\_new\_high**

**if \_\_name\_\_ == '\_\_main\_\_':**

**radius\_ratio = 0.5 # 圆形过滤器的半径：ratio \* w/2**

**img = cv.imread('../images/bb.jpg', cv2.IMREAD\_GRAYSCALE)**

**low\_freq\_part\_img, high\_freq\_part\_img = get\_low\_high\_f(img, radius\_ratio=radius\_ratio) # multi channel or single**

**plt.subplot(131), plt.imshow(img, 'gray'), plt.title('Original Image')**

**plt.axis('off')**

**plt.subplot(132), plt.imshow(low\_freq\_part\_img, 'gray'), plt.title('low\_freq\_img')**

**plt.axis('off')**

**plt.subplot(133), plt.imshow(high\_freq\_part\_img, 'gray'), plt.title('high\_freq\_img')**

**plt.axis('off')**

**plt.show()**